VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

AKCELERACE NEURONOVÉ SÍTĚ PRO JAZYKOVÉ MODELOVÁNÍ

NEURAL LANGUAGE MODEL ACCELERATION

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **DOMINIK LABAŠ**
AUTHOR

VEDOUCÍ PRÁCE **Ing. KAREL BENEŠ**
SUPERVISOR

BRNO 2018

Zadání bakalářské práce/20644/2017/xlabas00

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav počítačové grafiky a multimédií Akademický rok 2017/2018

Zadání bakalářské práce

Řešitel: **Labaš Dominik**
Obor: Informační technologie
Téma: **Akcelerace neuronové sítě pro jazykové modelování**
Neural Language Model Acceleration
Kategorie: Zpracování řeči a přirozeného jazyka

Pokyny:

1. Seznamte se s použitím neuronových sítí v jazykovém modelování
2. Prostudujte používané architektury dopředných sítí
3. Implementujte dopředný průchod neuronovou sítí
4. Navrhněte urychlení výpočtu
5. Implementujte ho a změřte reálné urychlení

Literatura:

- Y. Bengio, R. Ducharme, P. Vincent, C. Jauvin: A Neural Probabilistic Language Model; http://www.jmlr.org/papers/volume3/bengio03a/bengio03a.pdf
- Y. Huang, A. Sethy, B. Ramabhadran: Fast Neural Network Language Model Lookups at N-Gram Speeds; http://isca-speech.org/archive/Interspeech_2017/pdfs/0564.PDF

Podrobné závazné pokyny pro vypracování bakalářské práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva bakalářské práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap (20 až 30% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Beneš Karel, Ing.**, UPGM FIT VUT
Datum zadání: 1. listopadu 2017
Datum odevzdání: 16. května 2018

doc. Dr. Ing. Jan Černocký
vedoucí ústavu

Abstrakt

Táto práca sa zaoberá akceleráciou neurónovej siete pre jazykové modelovanie. Cieľom práce je optimalizovať model doprednej neurónovej siete. Pri urýchľovaní neurónovej siete sme využili zmenu aktivačnej funkcie, predpočítanie matíc pre výpočet skrytej vrstvy, implementáciu cache histórie modelu a odstránenie normalizácie. Model s najlepšími výsledkami bol zrýchlený o 75.3%.

Abstract

This work adresses the topic of neural language model acceleration. The aim of this work is to optimize model of a feed-forward neural network. In accelerating of the neural network we used a change of activation function, pre-calculation of matrices for calculationg the hidden layer, implementation of the model's history cache and unnormalized model. The best-performing model was accelerated by 75.3%.

Klíčová slova

Jazykové modelovanie, Jazykový model, Dopredná neurónová sieť, Akcelerácia neurónovej siete

Keywords

Language modeling, Language model, Neural Network, Neural network acceleration

Citace

LABAŠ, Dominik. *Akcelerace neuronové sítě pro jazykové modelování.* Brno, 2018. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Karel Beneš

Akcelerace neuronové sítě pro jazykové modelování

Prohlášení

Prehlasujem, že som túto bakalársku prácu vypracoval samostatne pod vedením pána Ing. karla Beneša. Uviedol som všetky literárne pramene a publikácie z ktorých som čerpal.

.....
Dominik Labaš
16. května 2018

Poděkování

Chcel by som poďakovať pánovi Ing. Karlovi Benešovi za jeho rady, skvelú pomoc a nekonečnú trpezlivosť pri vypracovávaní tejto práce.

Obsah

Kapitola 1

Úvod

Štatistické azykové modelovainie je proces rozdelenia pravdepodobností nad reťazcami slov. Samotný jazykový model je funkcia, ktorá tieto pravdepodobnosti dokáže vypočítať. V minulosti boli ako jazykové modely používane n-gram modely, ktoré sledovali počet výskytov jednotlivých slov z ktorých neskôr počítali pravdepodobnosť.

Pri využití n-gram modelov vznikali problémy, kde model bol využiteľný len pre dáta, ktoré boli podobné trénovacím dátam a taktiež vznikala takzvaná *Curse of Dimensionality*. *Curse of Dimensionality* je dôsledok vysokého počtu rôznych slov nachádzajúcich sa v trénovacích dátach, kde model pracuje so slovami ako s diskrétnymi jednotkami a nedokáže sa naučiť vzťahy, ktoré tieto slová nesú v prirodzenom jazyku. Z toho dôvodu rapídne narastá počet dimenzií priestoru v ktorom sa náš model pohybuje.

V roku 2003 bol navrhnutý spôsob, kde pri jazykovom modelovaní využijeme ako jazykový model neurónovú sieť [1]. Neurónová sieť je výpočtový model navrhnutý tak, aby sa trénovaním naučil vykonávať určitú funkciu. Neurónové sieťe využívame pri jazykovom modelovaní pre ich schopnosť učiť sa reprezentácie slov, ktoré pomáhajú s odstránením efektu *Curse of Dimensionality* a taktiež efektívne reprezentujú sémantické vlastnosti slov.

V tejto práci sa budeme venovať rýchlostiam neurónových sietí využívaných pri jazykovom modelovaní. Pri optimalizovaní modelu sa pokúsime zreplikovať štyri techniky pre urýchlenie výpočtu neurónovej siete [6]: zámena aktivačnej funkcie, implementácia jednoduchej histórie obsahujúcej predošlé hodnoty skrytej vrstvy, vytvorenie matíc pre výpočet skrytej vrstvy a odstránenie normalizácie modelu.

Kapitola 2

Neurónová sieť ako pravdepodobnostný jazykový model

Štatistické jazykové modelovanie pracuje s reťazcami slov v tvare $(w_1, w_2, ..., w_{N-1}, w_N)$ zkrátene (w_1^N), pričom jeho cieľ je vypočítať pravdepodobnosť $P(w_1^N)$. Túto pravdepodobnosť je možné faktorizovať na súčin podmienených pravdepodobností ako:

$$P(w_1^N) = P(w_1) \cdot P(w_2|w_1) \cdot ... \cdot P(w_N|w_1, w_2, ..., w_{N-1}) = \prod_{t=1}^{N} P(w_t|w_1^{t-1}) \tag{2.1}$$

Pravdepodobnosť $P(w_2|w_1)$ vyjadruje pravdepodobnosť výskytu slova w_2 ak predošlé slovo je práve w_1.

Úlohou jazykového modelovania je teda naučiť sa funkciu rozdelenia pravdepodobností f v tvare:

$$f(w_N, w_{N-1}, ..., w_1) = P(w_N|w_1^{N-1}) \tag{2.2}$$

Táto funkcia f reprezentuje jazykový model, ktorý dokáže tieto pravdepodobnosti počítať.

Perplexita jazykového modelu značí úspešnosť modelu priradiť vyššiu pravdepodobnosť tým dátam, ktoré sú často videné a gramaticky správne. Perplexitu modelu meriame na dátach, ktoré modelom neboli videné počas trénovania. Perplexitu je možné spočítať ako:

$$PPL = \frac{1}{\sqrt[N]{P(w_1^N)}} \tag{2.3}$$

V praxi potom platí, čím menšia perplexita, tým lepší model.

2.1 Existujúce riešenie jazykového modelu

Tvorba jazykového modelu bola v minulosti riešená pomocou takzvaných *n-gram* modelov [2]. Tie využívajú Markovské procesy n-tého rádu, teda pravdepodobnosť daného slova je určená podľa predošlých *n - 1* slov:

$$P(w_t|w_1^{t-1}) \approx P(w_t|w_{t-n+1}^{t-1}) \tag{2.4}$$

Teda funkciu jazykového modelu (2.2) môžme upraviť na tvar:

$$f(w_t, w_{t-1}, ..., w_{t-n+1}) = P(w_t|w_{t-n+1}^{t-1}) \tag{2.5}$$

Tieto modely potom sledujú skupiny slov o dĺžke n a ukladajú si počet výskytov, z ktorých je podom odvodená pravdepodobnosť. Najjednoduchší je *uni-gram* model ktorý sleduje len počet výskytov jednotlivých slov, teda:

$$f(w_t) = \frac{count(w_t)}{\sum_{i=1}^{N} count(w_i)} \tag{2.6}$$

Bi-gram model sleduje históriu o dĺžke jedného slova:

$$f(w_t, w_{t-1}) = \frac{count(w_{t-1}, w_t)}{count(w_{t-1})} \tag{2.7}$$

Z čoho môžeme odvodiť výpočet pre *n-gram* model ako:

$$f(w_{t-n+1}^{t}) = \frac{count(w_{t-n+1}^{t})}{count(w_{t-n+1}^{t-1})} \tag{2.8}$$

2.2 Problémy n-gram modelov

Jeden z najväčších problémov *n-gram* modelov je ten, že model sa dá využiť len pre dáta podobné dátam použitých pri trénovaní modelu. V prípade ak vyžadujeme od modelu pravdepodobnosť pre kombináciu histórie, ktorú ešte nevidel nastane delenie nulou. Tento problém bol riešený takzvanými *smoothing* metódami [3]. Medzi ktoré patrí napríklad pridanie hodnoty 1 pri každom výpočte:

$$f(w_{t-n+1}^{t}) = \frac{count(w_{t-n+1}^{t}) + 1}{count(w_{t-n+1}^{t-1}) + V} \tag{2.9}$$

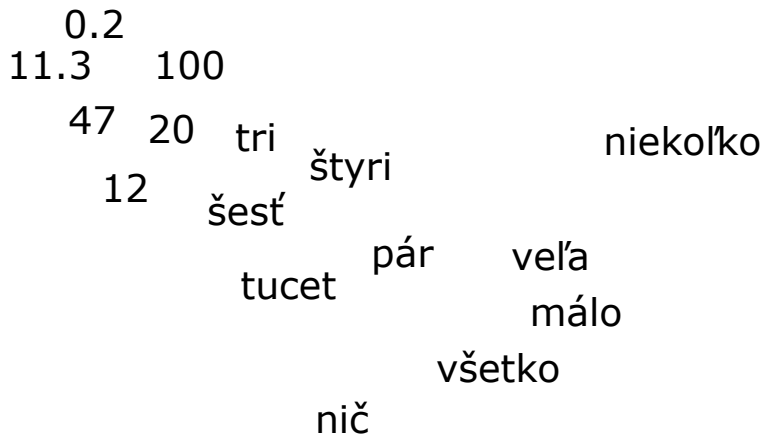
Ďalší možný spôsob je využitie *backoff* modelov, ktoré napríklad ak majú dostatok informácií využijú *tri-gram* inak *bi-gram*. *Interpolation* modely kombinujú výsledky viac *n-gram* modelov.

Uni-gram model reprezentuje slová ako jedno číslo vyjadrujúce počet jeho výskytou. V prípade, kedy pracujeme s *n-gram* modelami vyššieho rádu ako *uni-gram*, je potrebné si pamätať počet výskytov jedného slova pre každú možnú kombináciu, ktorá sa pred daným slovom mohla vyskytnúť. Všetky tieto hodnoty nemajú spoločné vlastnosti a preto sa počet dimenzií priestoru v ktorom sa náš model pohybuje zväčšuje. Ako dôsledok zväčšujúceho sa počtu dimenzií vzniká takzvaná *Curse of Dimensionality*. So zväčšujúcim sa počtom dimenzií sa vzdialenosť jednotlivých prvkov stáva podobná a zabraňuje prvkom aby sa rozložili do skupín so spoločnými vlastnosťami.

2.3 Neurónová sieť

Neurónová sieť je výpočetný model, ktorý sa pomocou trénovania učí vykonávať požadovanú funkciu. Výhoda neurónových sietí oproti *n-gram* modelom je v tom, že pracujú s

reprezentáciami slov, ktoré sa nachádzaju v spoločnom priestore. Tieto reprezentácie slov môžme využiť na popísanie vlastností slov, čo nám umožní v priestore vytvoriť takzvaný *clustering*. *Clustering* spôsobuje to, že slová ktoré majú podobné vlastnosti sa v priestore zhlukujú do skupín ako je možno vidieť na príklade v obrázku 2.1.



Obrázek 2.1: Príklad javu *clustering* zhlukujúci výrazy hodnôt do skupiny.

Pri počítaní pravdepodobností sa neurónová sieť učí funkciu jazykového modelu (2.5) pri ktorej namiesto slov w_i využíva reprezentácie slov:

$$\mathrm{f}(w_t, w_{t-1}, ..., w_{t-n+1}) = \mathrm{g}(w_t, \vec{C}(w_{t-n+1}), ..., \vec{C}(w_{t-1})) \tag{2.10}$$

Funkcia g predstavuje neurónovú sieť, ktorá počíta pravdepodobnosť slova w_t a vektor $\vec{C}(w_i)$ predstavuje reprezentácie slov.

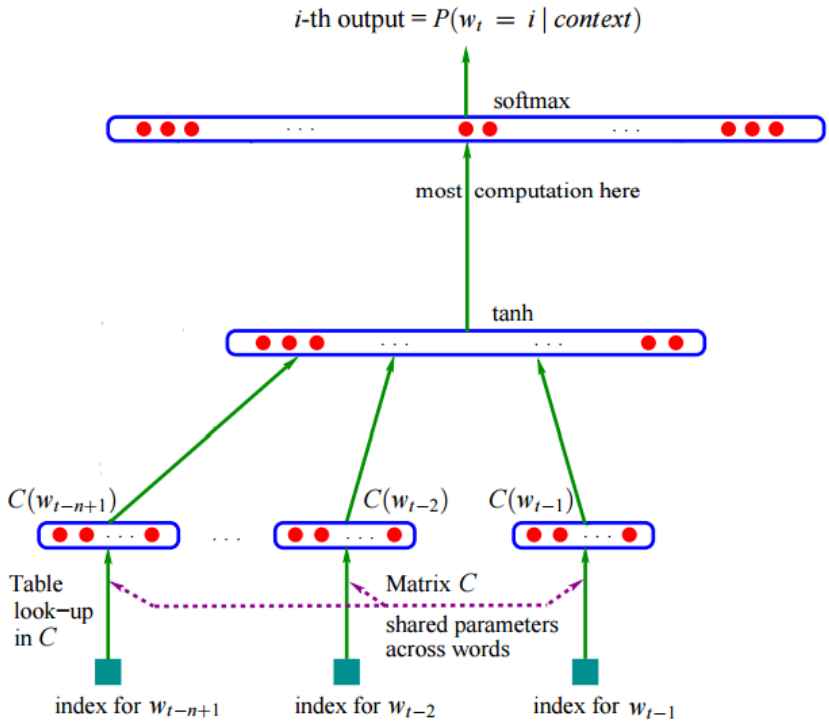
2.3.1 Reprezentácia vstupu a výstupu

Na vstupe modelu sa nachádzajú slová w_i kde $\forall w_i \in V$. Slovník V je konečná množina všetkých rôznych slov vyskytujúcich sa vo vstupných dátach. Tieto slová sú modelom mapované na vektory vlastností $\vec{C}(w_i)$. Vektory vlastností slúžia ako reprezentácie slov o veľkosti m a sú uložené v matic $\mathbf{C}$ s veľkosťou $|V| \times m$. Tieto reprezentácie slov sa model učí v priebehu trénovania súčasne s pravdepodobnostnou funkciou (2.5).

Obsah vstupnej vrstvy neurónovej siete je zreťazená sekvencia vektorov $\vec{c} = (\vec{C}(w_{t-n+1}), ..., \vec{C}(w_{t-1}))$. Túto sekvenciu ďalej model mapuje na rozdelenie pravdepodobnosti pre nasledujúce slovo w_t nad slovníkom V. Výsledkom je vektor pravdepodobností o veľkosti $|V|$ vyjadrujúci pravdepodobnosti (2.5) pre všetky slová slovníka V. Na výstupe modelu sa nachádza tento vektor $(P(w_1), P(w_2), ..., P(w_{|V|}))$ (obr. 2.2).

2.4 Dopredná neurónová sieť s jednou skrytou vrstvou

Táto neurónová sieť sa zkladá z troch vrstiev: vstupnej, skrytej a výstupnej. Pri zvolení dostatočnej dĺžky skrytej vrstvy táto sieť dokáže aproximovať funkciu (2.5). Každá z týchto vrstiev sa skladá z neurónov, kde každý neurón obsahuje jedno číslo, ktoré nazývame vnútorný potenciál neurónu. Počas priechodu neurónovou sieťou sa s hodnotami uloženými vo vrstvách pracuje ako s vektormi na ktoré sú aplikované váhy. Pri aplikovaní váh sú



Obrázek 2.2: Jazykový model, vstupmi sú indexy na slová zo slovníka V. Vektory vlastností $\vec{C}(w_i)$ sú zreťazené na vstupnej vrstve neurónovej siete predstavovanej funkciou g z (2.10). Výstupom je vektor rozdelenia pravdepodobností nad V. Prevzaté z [1]

.

nimi vynásobené hodnoty v predošlej vrstve, ktoré sú nasledovne spočítané do vnútorného potenciálu neurónu z nasledujúcej vrstvy.

Po namapovaní indexov slov na ich reprezentácie a vložení na vstupnú vrstvu siete sa na jednotlivé hodnoty aplikujú váhy ako:

$$\vec{h} = \varphi(\mathbf{H}\vec{c} + \vec{d}) \tag{2.11}$$

Vektor $\vec{h}$ predstavuje vektor s výstupom skrytej vrstvy. Vektor $\vec{d}$ obsahuje biase pre skrytú vrstvu, matica $\mathbf{H}$ (o veľkosti $|\vec{h}| \times |\vec{c}|$) obsahuje váhy pre prechod z vstupnej vrstvy do skrytej a vektor $\vec{c}$ obsahuje vstupnú vrstvu, kde sú zreťazené sekvencie vektorov $(\vec{C}(w_{t-n+1}), ..., \vec{C}(w_{t-1}))$. Funkcia φ predstavuje aplikačnú funkciu ktorá sa aplikuje na každý element skrytej vrstvy.

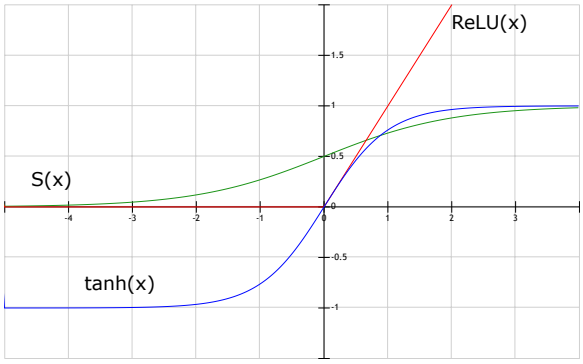
Ako aktivačná funkcia sa používajú funkcie ako logistická funkcia, hyperbolický tangens alebo funkcia ReLU (Rectified Linear Unit) (2.3).

Následne sú na skrytú vrstvu aplikované váhy pre prechod do výstupnej vrstvy:

$$\vec{y} = \mathbf{U}\vec{h} + \vec{b} \tag{2.12}$$

Vektor $\vec{y}$ (o veľkosti $|V|$) predstavuje skóre vypočítané neurónovou sieťou, vektor biasov $\vec{b}$ pre výstupnú vrstvu a maticou váh $\mathbf{U}$ (o veľkosti $|V| \times |\vec{h}|$) pre prechod zo skrytej do výstupnej vrstvy a $\vec{h}$ predstavuje vektor so skrytou vrstvou.

Na výstupnú vrstvu sa aplikuje funkcia softmax, ktorá zaisťuje normalizáciu modelu:



Obrázek 2.3: Porovnanie aktivačných funkcií pre neurónovú sieť. Funkcia S predstavuje logistickú funkciu, kde $S(x) = 1/(1 + e^{-x})$, funkcia $ReLU(x) = \max(0, x)$ a tanh $= (e^x - e^n - x)/(e^x + e^{-x})$.

$$P(w_t|w_{t-1}^{t-n+1}) = \text{softmax}(y) = \frac{e^{y_{wt}}}{\sum_{i=1}^{|V|} e^{y_{wi}}} \tag{2.13}$$

Táto funkcia zaisťuje, že súčet všetkých hodnôt na výstupe modelu bude 1 a tým pádom je možné s týmito hodnotami naďalej pracovať ako s pravdepodobnosťami.

Pri využití doprednej neurónovej siete perplexitu počítame ako geometrický priemer z $1/P(w_t|w_1^{t-1})$. Počas trénovania modelov sa ich úspešnosť meria pomocou hodnoty perplexity, kde menšia hodnota značí úspešnejší model. Tieto hodnoty sa získavajú priebežne z trénovacích dát a po dokončenom cykle cez trénovacie dáta z validačných dátach. Po dokončenom trénovaní sa perplexita vyráta pre testovacie dáta.

Kapitola 3

Optimalizácia modelu

Pri optimalizácii sa budeme pracovať s jazykovým modelom využívajúcim doprednú neurónovú sieť pre počítanie pravdepodobnosti z funkcie (2.5). Pre zrýchlenie nášho medlu využijeme štyri techniky: zámena aktivačnej funkcie, implementácia jednoduchej histórie obsahujúcej predošlé hodnoty skrytej vrstvy, vytvorenie matíc pre výpočet skrytej vrstvy a odstránenie normalizácie modelu [6].

3.1 Časová náročnosť modelu

Pri popise teoretickej náročnosti modelu si proces výpočtu neurónovej siete rozdelíme na niekoľko častí, ktoré budeme optimalizovať.

Aplikácia váh medzi vstupnou a skrytou vrstvou pozostáva z násobenia vektoru vstupnej vrstvy $\vec{c}$ s maticou obsahujúcou váhy $\mathbf{H}$. Veľkosť vektora $|\vec{c}| = m \times n$, kde m predstavuje veľkosť reprezentácií slov a n predstavuje počet slov spracovávaných modelom. Veľkosť matice $\mathbf{H}$ je $|\vec{h}| \times |\vec{c}|$, kde $|\vec{h}|$ je veľkosť skrytej vrstvy. Pre výpočet skrytej vrstvy je teda potreba $m \cdot n \cdot |\vec{h}|$ násobení. Po násobení sa k výslednému vektoru pripočítava vektor obsahujúci biase skrytej vrstvy, čo k náročnosti pridáva $|\vec{h}|$ sčítaní, tento krok je značne rýchlejší ako násobenie váh a preto ho môžme vynechať. Z toho určíme čas výpočtu ako:

$$T_h = m \cdot n \cdot |\vec{h}| \tag{3.1}$$

Po projekcii vstupnej vrstvy na skrytú vrstvu sa aplikuje aktivačná funkcia tanh(). čas potrebný pre aplikovanie aktivačnej funkcie na vektor o dĺžke x označíme ako $T_tanh(x)$. Teda čas pre výpočet funkcie tanh() pre skrytú vrstvu bude:

$$T_t = T_{tanh}(|\vec{h}|) \tag{3.2}$$

Po aplikácii aktivačnej funkcie nasleduje projekcia zo skrytej vrstvy na výstupnú vrstvu obsahujúcu skóre vypočítané neurónovou sieťou. Tento výpočet sa skladá z násobenia vektoru $\vec{h}$ s maticou váh pre výstupnú vrstvu $\mathbf{U}$ o veľkosti $|V| \times |\vec{h}|$. Teto krok zaberie $|\vec{h}| \cdot |V|$ násobení, pripočítavanie biasov zanedbávame.

$$T_o = |\vec{h}| \cdot |V| \tag{3.3}$$

Posledná časť modelu je výpočet funkcie softmax (2.13), ten si označíme podobne ako výpočet funkcie tanh(). Teda výpočet funkcie softmax pre vektor $\vec{y}$ ($|\vec{y}| = |V|$) bude:

$$T_s = T_{softmax}(|V|) \tag{3.4}$$

3.2 Jednotlivé optimalizácie modelu

Každá optimalizácia modelu bude cielená na istú časť modelu popísanú v predošlej kapitole. Cieľom optimalizácií je zmenšiť počet operácií vykonávaných modelom. Tento cieľ je dosiahnuteľný dvomi spôsobmi: úpravou výpočtu tak, aby bol nahradený rýchlejším výpočtom alebo do pamäte modelu uložiť hodnoty, ktoré umožnia urýchlenie modelu pri nasledujúcich výpočtoch.

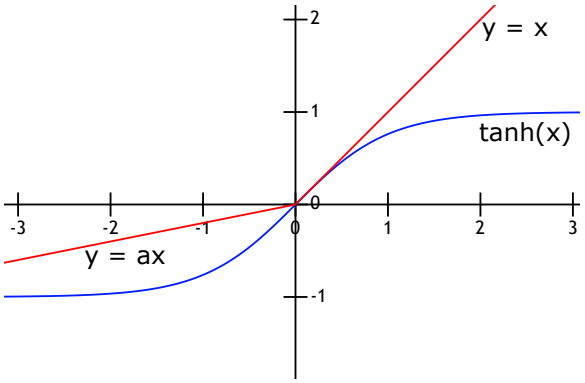
3.2.1 Zmena aktivačnej funkcie

Pre náš základný model bola zvolená aktivačná funkcia tanh(), ktorú pri prvej optimalizácii nahradíme funkciou PReLU() (Parametric Rectified Linear Unit) [10]. Funkcia PReLU() je podobná funkcii ReLU() (pozri 2.3), ktorá nenahrádza záporné čísla nulou ale násobí ich parametrom a, ktorý je uložený vo vektore o veľkosti skrytej vrstvy a model sa ho učí v priebehu trénovania. Funkcia PReLU() počíta výstup skrytej vrstvy ako:

$$\text{PReLU}(x) = max(0, x) + a \cdot min(0, x) \tag{3.5}$$

Vo výsledku nahradíme caš potrebný pre výpočet tanh() (3.2) časom pre výpočet maxima pre dva vektory o dlžke skrytej vrstvy $|\vec{h}|$. Tento čas môžme označiť ako:

$$T_t = 2 \cdot T_{max}(|\vec{h}|) \tag{3.6}$$



Obrázek 3.1: Graf aktivačných funkcií PReLU(x) a tanh(x).

3.2.2 Ukladanie histórie

Pri spracovaní textu jazykovým modelom sa často stáva, že sa reťazce slov opakujú. Z toho dôvodu budeme model optimalizovať pomocou implementovania jednoduchej histórie (cache). Táto história bude ukladať vektor s obsahom skrytej vrstvy $\vec{h}$ pre videnú históriu. V prípade keď budeme od modelu žiadať vypočítanie pravdepodobností pre modelom videný vstup bude skrytá vrstva načítaná z pamäte modelu. Ak čas potrebný pre výpočet skrytej vrstvy je $T_h + T_t$ po implementovaní histórie sa tento čas zmenší na $(1 - hit) \cdot (T_h + T_t)$, kde *hit* označuje pravdepodobnosť, že vstup modelu je uložený v histórii.

3.2.3 Predpočítanie matíc pre výpočet skrytej vrstvy

Pretože po trénovaní sa výsledky mapovania indexov na reprezentácie slov a váhy pre prechod do skrytej vrstvy ďalej nemenia, je možné si skrytú vrstvu predpočítať. Toto docielime tým, že si vytvoríme matice $\mathbf{M}_j$ o veľkosti $|V| \times |\vec{h}|$ pre všetky slová zo slovníku V a všetky možné pozície vo vektore $\vec{c}$. Pri počítaní skrytej vrstvy potom stačí sčítať vektory z matíc pre zadané indexy a v modeli pokračovať od aplikačnej funkcie. [4]

$$\vec{h} = \tanh(\sum_{j=0}^{n-1} \mathbf{M}_j[w_j]) \tag{3.7}$$

Výraz $\boldsymbol{M}_j[w_j]$ značí indexáciu na pozíciu prvku w_j. Biase pre skrytú vrstvu pripočítavame len v jednej matici. Týmto spôsobom ušetríme čas potrebný pre výpočet funkcie (2.11) a výsledný čas bude:

$$T_h = n \cdot |\vec{h}| \tag{3.8}$$

3.2.4 Odstránenie normalizácie modelu

Po spočítaní výstupnej vrstvy modelom je aplikovaná funkcia *softmax* (2.13), ktorá zaručuje to, že suma celej vrstvy bude 1 tak, aby výsledok bol použiteľný ako pravdepodobnosť. Pri počítaní funkcie *softmax* nás najviac spomaluje vypočítavanie výrazu $\sum_{i=1}^{|V|} e^{y_{wi}}$. Pre urýchlenie výpočtu pravdepodobností v tomto prípade využijeme *Noise contrastive estimation* [5], touto zmenou odstránime normalizáciu modelu tak, že už nebude potrebné počítať sumu pre všetky hodnoty výstupnej vrstvy, ale iba jedného slova, ktorého pravdepodobnosť chceme počítať a k takzvaných *noise* hodnôt. Vo výsledku nahradíme funkciu *softmax* funkciou na výpočet pravdepodobností v tvare:

$$P(w_t|w_{t-1}^{t-n+1}) \approx \frac{e^{y_{wt}}}{e^{y_{wt}} + \sum_{i=1}^{k} e^{y_{wi}}} \tag{3.9}$$

Týmto výpočtom nebudeme naďalej počítať rozdelenie pravdepodobnosti cez celú sadu slov zo slovníka V, ale len pravdepodobnosť pre nami vybraté slovo y_{wt}. Pri využití tejto techinky dokážeme urýchliť nie len výpočet výsledného modelu ale dosiahneme aj zrýchlenie trénovania modelu. Účinok tejto techniky sa zvyšuje pri spracovaní textov s vysokou veľkosťou slovníka V. Vo vzorci výpočtu pravdepodobností pre nenormalizovaný model (4.7) je možné sledovať stav, kde ak sa veľkosť hodnoty k blíži veľkosti slovníku $|V|$ sa výpočet začne približovať výpočtu pre normalizovaný model (2.13). Týmto trénovaním dosiahneme, že skóre vypočítané na výstupnej vrstve budú mať hodnoty v rozsahu [0, 1] a ich suma sa bude približovať hodnote 1. Tým pádom bude možné využívať ako pravdepodobnosti samotné skóre počítané modelom, teda pri evaluácii nebude nutné využívať žiadnu normalizačnú funkciu a čas $T_s = 0$.

3.3 Kombinácia optimalizácií

Navrhnuté optimalizácie sú spolu kombinovateľné podľa časti, ktorú upravujú. Odstránenie normalizácie je kombinovateľné s každou inou optimalizáciu, pretože sa ako jediná zaoberá výpočtom nad výstupnou vrstvou.

Kombinácia s optimalizáciami s ukladaním histórie a predpočítanými maticami ovplyvňujú výpočet skrytej vrstvy a preto ich kombinácia nebude veľmi výhodná. Taktiež kombinácia histórie cache a aktivačnej funkcie PReLU() sa navzájom prekrývajú a z toho dôvodu by bolo zrýchlenie aktivačnou funkciou zmenšené na $(1 - hit) \cdot 2 \cdot T_{max}(|\vec{h}|)$.

Pri využití aplikačnej funkcie PReLU() a predpočítaním matíc pre výpočet skrytej vrstvy sú veľmi vhodné na kombináciu, pretože optimalizácia aktivačnej funkcie priamo nadväzuje na optimalizáciu pomocou predpočítaných matíc. časová zložitosť modelu by sa zmenila z $m \cdot n \cdot |\vec{h}| + T_{tanh}(|\vec{h}|)$ na $n \cdot |\vec{h}| + 2 \cdot T_{max}(|\vec{h}|)$.

Vo výsledku budú optimalizácie kombinované v troch smeroch: nenormalizovaný model a história cache, nenormalizovaný model spolu s aplikačnou funkciou a predpočítanými maticami. Posledná odporúčaná kombinácia je aplikačná funkcia PReLU() a predpočítané matice, pretože táto optimalizácia nezmení tvar výstupu modelu.

Kapitola 4

Implementácia

Jazykový model je implementovaný v jazyku Python 2.7 za použitia knižnice PyTorch [9] určenému k práci s neurónovými sieťami.

Základom implementácie sú dva skripty určené pre prácu s navrhnutými modelmi. Prvý na trénovanie modelu, kde sa úspešnosť modelu pozoruje pomocou vypočítavania straty. Druhý pre sledovanie času pri priechode cez model využitím profileru z knižnice PyTorch.

Vstupom prvého skriptu pre trénovanie sú tri rôzne texty určené na trénovanie, validáciu a testovanie modelu.

Výstupom tohto skriptu je natrénovaný model a štatistiky o priebehu trénovania, ktoré sú vypísané na štandardný výstup.

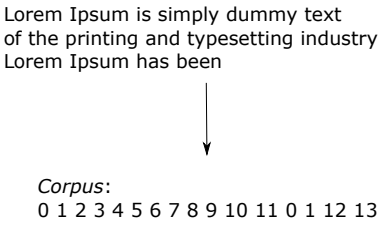
Druhý skript využíva nami natrénovaný model a pomocou testovacieho textu sleduje časovú náročnosť modelu, ktorú vypisuje na štandardný výstup. Informácie získavame pomocou funkcie profiler z knižníc PyTorch a po priechode cez celé dáta získame štatistiky ako najmenší, najväčší a priemerný čas priechodu cez model.

Pred tým, ako naše skripty začnú pracovať so samotným modelom, sú spracované vstupné texty. Zo vstupných textov sú vytvorené objekty *Corpus* a *Dictionary* (obrázok 4.1). Objekt *Corpus* obsahuje vstupný text konvertovaný do indexov typu *LongTensor*, tieto indexy určujú pozíciu daného slova v objekte *Dictionary*, ktorý obsahuje každé unikátne slovo vyskytujúce sa vo vstupných textoch. Následne sú indexy v *Corpus* rozdelené pomocou funkcie `batchify()` (obrázok 4.2) na niekoľko stĺpcov, vďaka ktorým môže model spracovávať viacero vstupov.

Vstup modelu je pole *input* obsahujúce indexy z objektu *Corpus* (obrázok 4.1) reprezentujúce slová zo spracovávaného textu. Pole *input* sa získava pomocou funkcioe `getbatch()`, ktorá taktiež vracia indexy na nasledujúce slová označené ako *target* (obrázok 4.3). Indexy *targets* sú využívané pre výpočet perplexity (2.3).

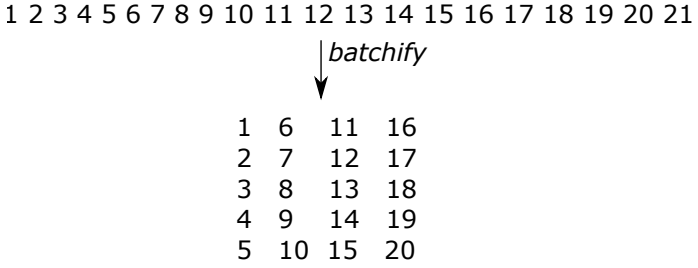
4.1 Tvorba modelu

Model je vytvorený pomocou knižnice PyTorch triedou *torch.nn.Module*. Základný model je navrhnutý ako dopredná neurónuvá sieť (obrázok 4.4), ktorá implementuje rovnicu (2.10). Výpočet modelu je implementovaný v metóde *model.forward()*.



Dictionary	
Word to index	*index to word*
Lorem : 0	Lorem
Ipsum : 1	Ipsum
is : 2	is
simply : 3	simply
dummy : 4	dummy
text : 5	text
of : 6	of
the : 7	the
printing : 8	printing
and : 9	and
typesetting : 10	typesetting
industry : 11	industry
has : 12	has
been : 13	been

Obrázek 4.1: Ukážka spracovania vstupných dát pomocou na objekty *Corpus* a *Dictionary*.

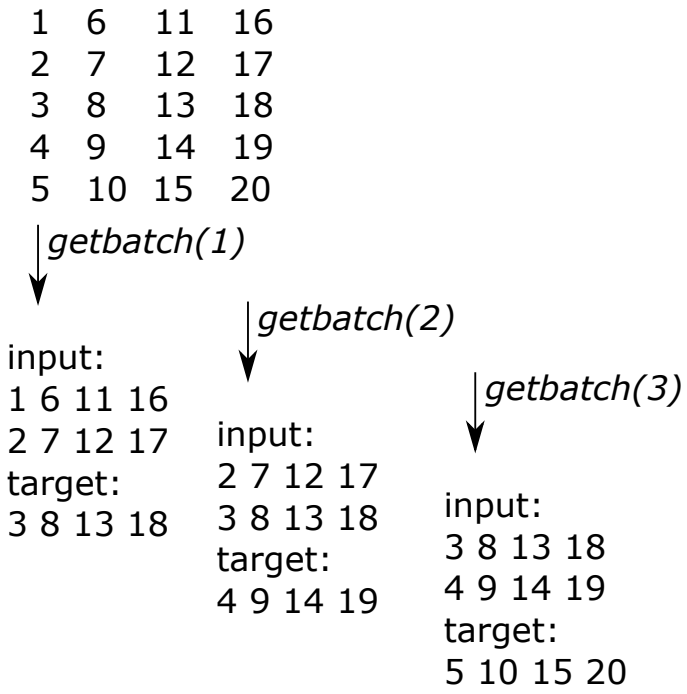


Obrázek 4.2: Ukážka spracovania objektu *Corpus* pomocou funkcie `batchify()`.

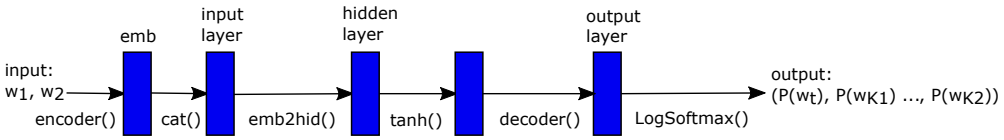
Listing 4.1: Základný tvar metódy model.forward()

```
def forward(self, input):
        emb = self.encoder(input)

        in_layer = torch.cat(torch.split(emb, 1), 2)
        in_layer = in_layer.view(self.batch_size, -1)

        hid_layer = self.emb2hid(in_layer)
        hid_layer = torch.nn.functional.tanh(hid_layer)

        out_layer = self.decoder(hid_layer)
        prob = torch.nn.LogSoftmax(out_layer)
        return prob
```

Na výstupe modelu sa nachádza vektor rozdelenia pravdepodobností $(P(w_1), P(w_2), ..., P(w_{|V|}))$ nad zoznamom slov z objektu *Dictionary*.



Obrázek 4.3: Ukážka funkcie `getbatch()`, `ktorá z dát vyberá vstup modelu pre aktuálny krok`.



Obrázek 4.4: Návrh baseline modelu, porovnaj s 4.1.

4.2 Optimalizácie modelu

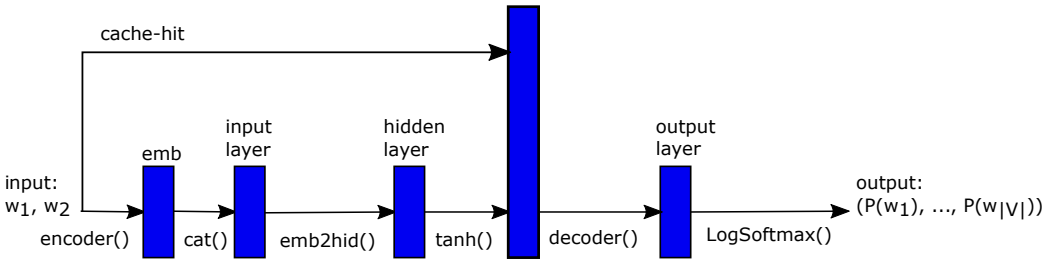
Pri optimalizácii som upravovali základný model, kde pre dosiahnutie rýchlejšieho výpočtu budeme upravovať a nahradzovať určité časti metódy `forward()`.

V prvom kroku bude mojím cieľom zrýchlenie výpočtu aktivačnej funkcie. Toho je možné dosiahnuť nahradením aktivačnej funkcie `tanh`() funkciou `PReLU`() [10], ktorej implementácia je zahrnutá v knižniciach PyTorchu.

- hid_layer = torch.nn.functional.tanh()
+ hid_layer = torch.nn.PReLU()

Tento krok nezmení využitie modelu a preto je možné s modelom pracovať pomocou skriptov rovnako ako so základným modelom. Model je potreba znova natrénovať z toho dôvodu, že výpočet pomocou funkcie `tanh`() naučí model pracovať s číslami v rozsahu [-1, 1] a funkcia `PReLU`() dáva výsledok v reálnych číslach.

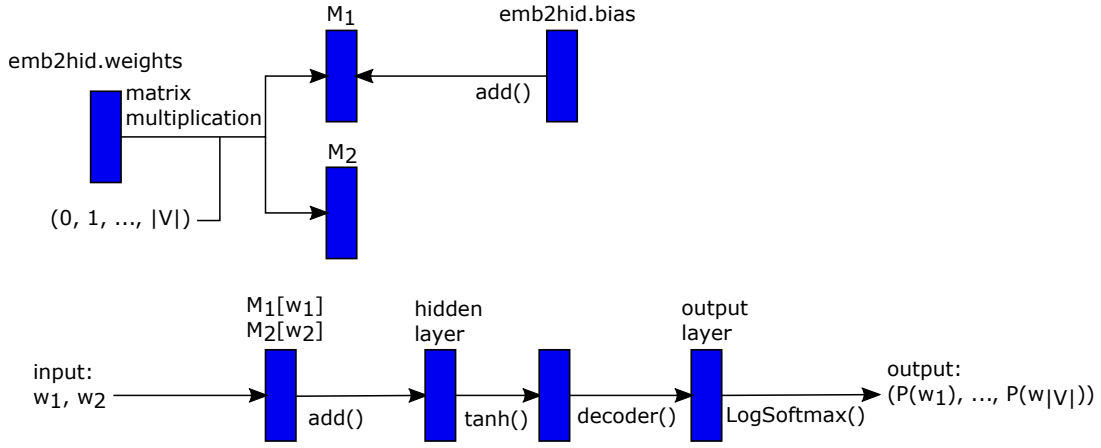
Druhá optimalizácia sa zaoberá rýchlejším výpočtom skrytej vrstvy pomocou ukladania stavu skrytej vrstvy do pamäte modelu. Táto hodnota je ukladaná spoločne s príslušným vstupom modelu. Následne je implementovaná jednoduchý cyklus pre sekvenčné vyhľadávanie vo videnej histórie modelu. Tento cyklus má za úlohu nájsť zhodu (cache-hit) medzi vstupom modelu a dátami uloženými v histórii. Po potvrdení zhody načítame hodnotu skrytej vrstvy s ktorou model následne pracuje. Postup ukladania do histórie je implementovaný metódou FIFO.



Obrázek 4.5: Model využívajúci ukladanie skrytej vrstvy do pamäte modelu.

Pre zvýšenie pravdepodobnosti, že nastane cache-hit nevyužívame funkciu `batchify()`. Túto optimalizáciu využívame len pri evaluačnom využití modelu.

Úlohou tretej optimalizácie je dosiahnuť rýchlejšieho výpočtu skrytej vrstvy pred aplikovaním aktivačnej funkcie. Do modelu budú uložené matice M_j (obrázok 4.6), ktoré obsahujú hodnoty reprezentácií slov vynásobené váhami pre prechod zo vstupnej do skrytej vrstvy. Biase sú zahrnuté tým spôsobom, že sú pripočítané do matice M_1. Pri priechode cez model sí matice M_j indexované podľa vstupu modelu, tieto hodnoty sú nálsedne spočítané do skrytej vrstvy na ktorú je nasledovne aplikovaná aktivačná funkcia (obrázok 4.6).



Obrázek 4.6: Model využívajúci ukladanie skrytej vrstvy do pamäte modelu.

Matice M_j sú vytvorené až po úspešnom natrénovaní modela a vplyv optimalizácie sa prejaví až pri evaluačnom využití modela.

4.3 Nenormalizovaný model

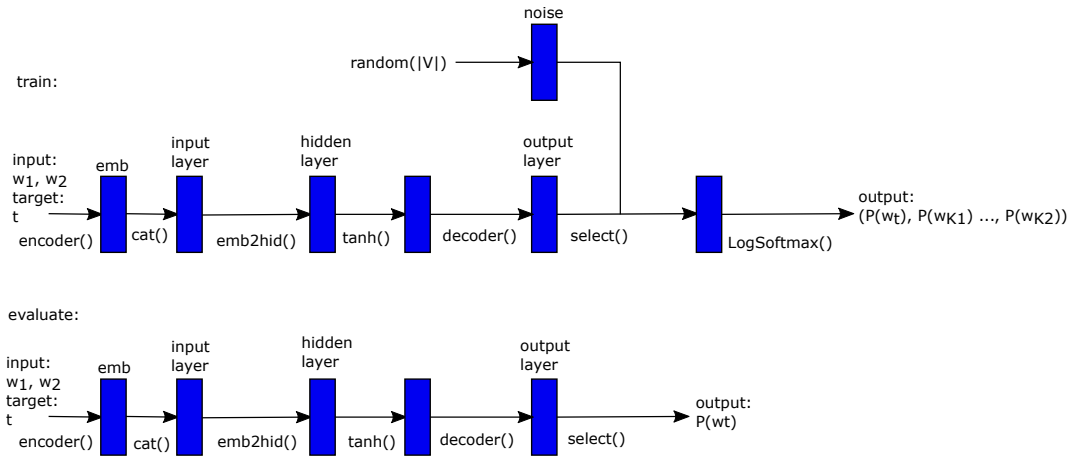
Vstup nenormalizovaného modelu sa rozšíri o index slova *target*, pre ktoré počítame pradepodobnosť.

Výpočet metódy `forward()` je rozdelený na dve časti: jednu pre trénovanie a druhú pre evaluáciu. Na začiatku modelu postupujeme rovnako ako pri baseline modeli 4.4.

Pri trénovaní bude po vypočítaní výstupnej vrstvy rovnakým spôsobom ako pri baseline modeli (obrázok 4.4) modelom generovaných K hodnôt. Tieto hodnoty sú generované z uniformného rozloženia *(0, 1, ..., |V|-1)*, ktoré budú využité ako *noise* (contrasive estimations) hodnoty.

Model následne indexuje výstupnú vrstvu tak, aby vytvoril vektor obsahujúce skóre pre požadovaný *target* nasledovaný skórami pre *noise* hodnoty. Na tento vektor aplikujeme funkciu `softmax` podľa vzorca (4.7). Výsledné hodnoty v tomto vektore sú výstup modelu, kde na indexe 0 sa nachádza nami požadovaná pravdepodobnosť.

Pri evaluačnom priechode modelom sa po vypočítaní výstupnej vrstvy indexuje skóre pre *target*, ktoré je v rozsahu [0, 1] a toto skóre nám slúži ako pravdepodobnosť $P(w_t)$. Táto hodnota sa následne nachádza na výstupe modelu.



Obrázek 4.7: Model využívajúci nenormalizovaný model. Prvá časť *train* sa využíva pri trénovaní modela, časť *evaluate* je využívaná modelom po úspešnom natrénovaní.

Kapitola 5

Experimenty

Cieľom experimentov je zmerať zrýchlenie dosahované mnou implementovanými modelmi, tie budem porovnávať na základe času potrebného pre výpočet metódy *forward*. Preto bude potrebné si zvoliť jeden model, ktorý bude slúžiť ako základný (baseline) model pre porovnanie úspešnosti optimalizácií modelov (4.1).

5.1 Vstupné dáta modelu, počiatočná konfigurácia

Ako vstup pre trénovanie a testovanie modelu sú využívané dáta *Penn Treebank* [8], jedná sa o text rozdelený medzi trénovacie, validačné a testovacie dáta o veľkosti:

Tabulka 5.1: Veľkosť vstupných dát.

Dáta	Slov
trénovacie	879196
validačné	69671
testovacie	78039

Texty sú upravené tak, že počet rôznych slov vyskytujúci sa v dátach je presne 10k. Toho je dosiahnuté tým, že slová, ktoré sa nevyskytujú v texte často sú nahradené jedným výrazom označujúcim slovo ako neznáme. Z dát sú skriptom odstránené konce riadkov tak, aby bol vytvorený ucelený text. Trénovacie dáta sa využívajú na trénovanie modelu. Validačné dáta slúžia k priebežnému meraniu perplexity po každej trénovacej epoche, kde sa dá sledovať prípadné pretrénovanie modelu. Testovacie dáta sa využívajú na konci trénovania pre zmeranie výslednej perplexity a pri meraní časovej náročnosti modelu.

Výpočet modelu je vykonávaný procesorom Intel® Core™ i5-6200U[1] s dvomi jadrami o frekvencii 2.30 GHz. Operačný systém Ubuntu 14.04. Využitá bola knižnica PyTorch, verzia 3.1. Pre zaistenie rovnakých podmienok pri meraní časovej náročnosti modelov bol ako jediný program spustená príkazová riadka, kde boli merané časové zložitosti modelu.

[1] https://ark.intel.com/products/88193/Intel-Core-i5-6200U-Processor-3M-Cache-up-to-2_80-GHz

5.2 Perplexita a rýchlosť baseline modelu

Prvým krokom experimentov bolo porovnať úspešnosť modelov na základe hodnoty perplexity z testovacích dát, pričom model s menšou perplexitou je úspešnejší ako model s väčšou perplexitou.

Tabulka 5.2: Porovnanie úspešností modelov. Baseline model je označený ako tanh, veľkosť značí veľkosť skrytej vrstvy a dĺžku reprezentácií slov. Vstup značí dĺžku vstupného reťazca modelu.

Model	Veľkosť	Vstup	Perplexita	Vstup	Perplexita
tanh	50	2	173.23	4	163.09
tanh	150	2	155.35	4	148.03
tanh	250	2	159.39	4	159.15
PReLU	150	2	164.21	4	162.84
história	150	2	155.35	4	148.03
projekcia	150	2	155.35	4	148.03
nenorm	150	2	169.43	4	168.67

Z výsledkov z tabuľky 5.2 môžeme vidieť, že úspešnosť modelu stúpa pri zväčšení vstupného reťazca. Najúspešnejší baseline model mal veľkosť skrytej vrstvy 150 neurónov. Perplexita baseline modelu, modelu využívajúceho históriu a modelu využívajúceho predpočítané matice je rovnaká a to z toho dôvodu, že ich optimalizácia nieje pri trénovaní využívaná. Perplexita pri využití funkcie `PReLU()` klesá a model je neúspešnejší. Podobne perplexita klesá aj pri odstránení normalizácie a to z toho dôvodu, že pri trénovaní používame len časť z modelom vypočítaných skór.

Rýchlosti modelov meriame počas jedného evaluačného priechodu cez testovacie dáta. Počas merania sledujeme čas strávený výpočtom metódy `model.forward()`. Modely následne porovnávame podľa najmenšieho, najväčšieho a priemerného nameraného času.

Na začiatok som porovnali baseline modely s rôznou veľkosťou skrytej vrstvy a dĺžky vstupného reťazca, kde sledujem ich vplyv na čas potrebný pre výpočet modelu. Z tabuľky 5.3 môžme vidieť, že čas potrebný na výpočet modelu stúpa s veľkosťou modelu.

Tabulka 5.3: Porovnanie rýchlostí základných modelov s rôznymi veľkosťami skrytej vrstvy a vstupného reťazca.

Model	Veľkosť	Vstup	Min	Max	Avg
tanh	50	2	4.72 ms	4.88 ms	4.75 ms
tanh	150	2	5.55 ms	5.81 ms	5.60 ms
tanh	250	2	6.27 ms	6.54 ms	6.30 ms
tanh	50	4	4.74 ms	4.93 ms	4.79 ms
tanh	150	4	5.58 ms	5.87 ms	5.65 ms
tanh	250	4	6.39 ms	6.61 ms	6.40 ms

Ako štandardnú veľkosť modelov pre experimentovanie s optimalizovanými modelami som vybrali veľkosť 150 so vstupným reťazcom o dĺžke 2 slov, pre ktorú ďalej pozorujem detailnú časovú záťaž modelu získanú pomocou profileru.

Tabulka 5.4: Ukážka výstupu z profileru znázorňujúci časuvú náročnosť jednotlivých krokov pre baseline model tanh 150.

Funkcia	Čas
embedding	145,2 μs
cat	44,5 μs
addmm	77,8 μs
tanh	147,7 μs
addmm	1530,9 μs
logsoftmax	4002,0 μs

Na výstupe profileru (tabuľka 5.4) môžme sledovať časové náročnosti jednotlivých častí modelu, kde normalizácia pomocou funkcie `LogSoftmax` jednoznačne zaberá najväčší čas z výpočtu. Výpočet aktivačnej funkcie a skrytej vrstvy zaberá iba zlomok z celkového času, teda zrýchlenie ich optimalizácií nebude mať na celkový výpočet výrazný vplyv.

5.3 Jednotlivé optimalizácie

Prvou optimalizáciou bolo nahradenie aktivačnej funkcie `tanh`() funkciou `PReLU`(), ako prvú časť experimentu som sledoval čas pre výpočet samostatnej aktivačnej funkcie, kde je možné tieto funkcie medzi sebou ľahko porovnať.

Tabulka 5.5: Rýchlosti výpočtu aktivačnej funkcie.

Aktivačná f	Avg
tanh()	147.7 μs
PReLU()	55.7 μs

V tabuľke 5.5 môžme pozorovať zhruba 3-násobné zrýchlenie výpočtu aktivačnej funkcie.

Následne som pozoroval štatistiky časov potrebných pre celkový výpočet modelov, kde je možné modely medzi sebou porovnať a určiť zrýchlenie. Modely použité na porovnávanie majú veľkosť 150 a využívajú vstupnú sekvenciu slov o dĺžke 2 slov.

Výsledok experimentu v tabuľke 5.6 ukázal 3.75% zrýchlenie celého modelu pomocou využitia aktivačnej funkcie `PReLU`().

Ako ďalší experiment som pozoroval zmenu pri využití implementovanej histórie modelu. Modely tohto experimentu nevyužívajú funkciu `getbatch()`, teda pri jednom priechode metódy `forward()` je počítaná pravdepodobnosť len pre jeden vstupný reťazec. Pri testovaní som využil modely o veľkosti 150 s vstupným reťazcom o dĺžke 2 slov, využité histórie majú

Tabulka 5.6: Celková rýchlosť modelov.

Model	Min	Max	Avg
tanh()	5.55 ms	5.81 ms	5.60 ms
PReLU()	5.37 ms	5.41 ms	5.39 ms

dĺžku 150, 300 a 500 naposledy videných vstupov.

Tabulka 5.7: Porovnanie rýchlostí modelov využívajúce ukladanie histórie (cache) s rôznou dĺžkou histórie a baseline modelu.

Model	Min	Max	Avg
baseline	0.68 ms	1.04 ms	0.72 ms
cache 150	0.54 ms	1.61 ms	1.01 ms
cache 300	0.54 ms	1.73 ms	1.04 ms
cache 500	0.54 ms	1.96 ms	1.17 ms

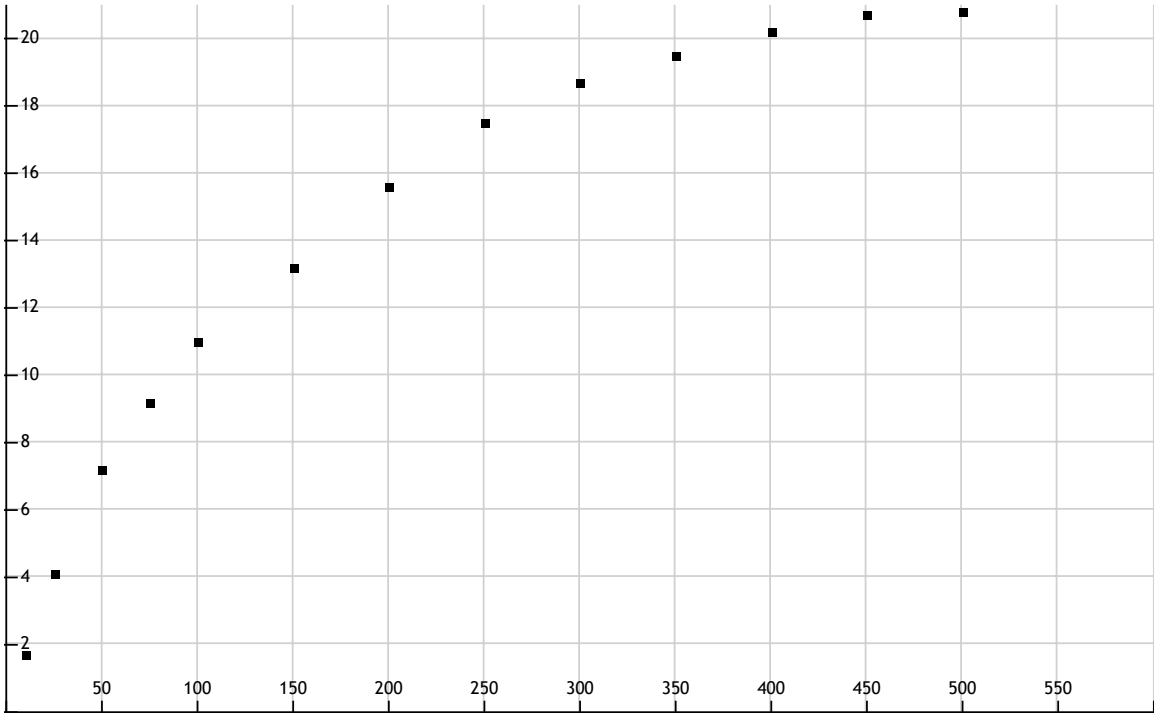
V tabuľke 5.7 výsledkov môžme v stĺpci *Min* pozorovať úspešné zrýchlenie modelu v prípade, ak sa vstup nachádza vo videnej histórii a skrytá vrstva je načítaná z histórie. V stĺpci *Max* vidíme opačný prípad, kedy sa vstup v histórii nenachádza a dochádza k nežiadúcemu spomaleniu modelu z dôvodu zdĺhavého cyklu, ktorý hľadá zhodu medzi históriou a vstupom. Výsledok experimentov ukazuje zhruba 20.5% zrýchlenie výpočtu pri úspešnom načítaní skrytej vrstvy z histórie. Ak sa daný vstup v histórii nevyskytuje pozorujeme spomalenie modelu o 54-88%. Výsledný priemerný čas výpočtu modelu dokazuje, že v najlepšom prípade naše riešenie spomaľuje model o 40%. Taktiež môžme sledovať, že pri rastúcej veľkosti histórie stúpa aj spomalenie modelu.

Tento problém by bolo možné riešiť pomocou implementácie v jazyku C z dôvodou menej časovo náročného indexovania a priebehu cyklu cez históriu. Taktiež môžme očakávať lepšie výsledky pri využití väčších modelov, ktorých výpočet pre dosiahnutie skrytej vrstvy vyžaduje dlhší čas.

Tabulka 5.8: Tabuľka pravdepodobnosti, že sa vstup modelu bude nachádzať v histórii modelu podľa jej dĺžky.

Model f	Úspešnosť
cache 150	13.2 %
cache 300	18.7 %
cache 500	20.8 %

Následne môžme pozorovať pravdepodobnosť úspechu vyhľadávania v histórii modelu (tabuľka 5.8), ktorej rast sa postupne spomaľoval. Túto skutočnosť môžme pozorovať aj na priemernom čase výpočtu, ktorý stúpal aj napriek zväčšovaniu veľkosti histórie.



Obrázek 5.1: Graf pravdepodobnosti úspechu histórie podľa veľkosti histórie. Osa x obsahuje počet vstupov uložených v histórii, osa y určuje pravdepodobnosť v %, že sa vstup modelu nachádza v histórii modelu.

Pri využití predpočítaných matíc som sledoval čas potrebný pre výpočet skrytej vrstvy pred aplikáciou aktivačnej funkcie. Tento čas som pozoroval pomocou výstupu z profileru, kde som spočítal súčet jednotlivých časov od začatí výpočtu modelu až po aplikáciu váh pre prechod na skrytú vrstvu.

Tabulka 5.9: Rýchlosť vypočítania skrytej vrstvy modelu.

Model	Vstup	Čas na získanie skrytej vrstvy
baseline	2	267.5 μs
projekcia	2	100.2 μs
baseline	4	311.2 μs
projekcia	4	133.4 μs

V tabuľke 5.9 pozorujeme výpočet skrytej vrstvy, ktorý je zhruba 2.5krát rýchlejší ako s baseline modelom. Ako ďalší krok som pozoroval vplyv tejto optimalizácie na celkový čas výpoču modelu.

V tabuľke 5.10 môžme pozorovať pre model o veľkosti 150 s vstupným reťazcom o dĺžke 2 slov zrýchlenie o 3.9%. Pre model s vstupným reťazcom o dĺžke 4 slov zrýchlenie o 2.8%. Z tohto experimentu vidíme, že efektivita využitia predpočítaných matíc klesá so zväčšovaním vstupného reťazca, čo zväčšuje potrebný počet uložených matíc.

Tabulka 5.10: Celková rýchlosť modelu podľa typu získavania skrytej vrstvy.

Model	Vstup	Min	Max	Avg
baseline	2	5.55 ms	5.81 ms	5.60 ms
projekcia	2	5.37 ms	5.39 ms	5.38 ms
baseline	4	5.58 ms	5.87 ms	5.65 ms
projekcia	4	5.46 ms	5.50 ms	5.49 ms

Ako poslednú časť samostatných optimalizácií som pozoroval modely, ktoré využívajú nenormalizovaný model. Tu smo pozorovali rýchlosť výpočtu funkcie `LogSoftmax`. Pre využitie nenormalizovaného modelu bolo potrebné zvoliť si počet generovaných *noise* hodnôt. V mojom prípade generujem 1000 *noise* hodnôt, čo je 10násobne menej honôt spracovávaných funkciou *softmax* ako pri normalizovanom modeli a z toho dôvodu môžem očakávať 10-násobné zrýchlenie. Tento experiment je rozdelený na dve časti, kde prvá časť skúma rýchlosť modelu počas trénovania a druhá po natrénovaní modelu.

Tabulka 5.11: Rýchlosť vypočítania funkcie `LogSoftmax`.

Model	Vstup	Čas výpočtu logsoftmax
baseline	2	4002.0 μs
nenorm	2	862.3 μs
baseline	4	4002.0 μs
nenorm	4	862.3 μs

Po odstránení normalizácie môžme v tabuľke 5.11 sledovať rýchlosť výpočtu funkcie `LogSoftmax` počas trénovania urýchlenú o 78%. Pôvodne sme mohli očakávať 10násobné zrýchlenie pri výpočte, ktoré skutočne nastalo a výpočet samotnej funkcie `LogSoftmax` sa blížil hodnote 400 μs, no pribudla záťaž indexácie a generácie *noise* hodnôt ktoré spôsobili, že konečné zrýchlenie záverečnej časti modelu bolo iba 5-násobné.

Tabulka 5.12: Celková rýchlosť modelu pri trénovaní podľa jeho normalizácie.

Model	Vstup	Min	Max	Avg
baseline	2	5.55 ms	5.81 ms	5.60 ms
nenorm	2	2.75 ms	2.79 ms	2.76 ms
baseline	4	5.58 ms	5.87 ms	5.65 ms
nenorm	4	2.78 ms	2.90 ms	2.80 ms

Urýchlenie výpočtu celého modelu v tabuľke 5.12 o veľkosti 150 a dĺžke vstupného reťazca 2 slov pri trénovaní bolo 50.7%. Pri dĺžke vstupného reťazca 4 slov bolo zrýchlenie 50,4%. Z toho môžme vidieť, že so zväčšujúcim sa modelom efektivita tejto optimalizácie na celý model klesá z toho dôvodu, že funkcia `LogSoftmax` zaberá menšiu časť celkového času.

V druhej časti som pozoroval efektivitu odstránenia normalizácie po natrénovaní modelu, kde došlo k úpnému odstráneniu funkcie `LogSoftmax`.

Tabulka 5.13: Rýchlosť vypočítania normalizácie pred a po jej odstránení.

Model	Vstup	Čas výpočtu logsoftmax
baseline	2	4002.0 μs
nenorm	2	14.7 μs
baseline	4	4002.0 μs
nenorm	4	14.7 μs

Výsledky po odstránení funkcie `LogSoftmax` môžme vidieť v tabuľke 5.13. Čas po odstránení funkcie nieje nulový z dôvodu potreby indexovať požadovanú pravdepodobnosť. Následne som pozoroval vplyv odstránenia tejto funkcie na celkový výpočet modelu.

Tabulka 5.14: Celková rýchlosť modelu podľa jeho normalizácie.

Model	Vstup	Min	Max	Avg
baseline	2	5.55 ms	5.81 ms	5.60 ms
nenorm	2	1.60 ms	1.74 ms	1.62 ms
baseline	4	5.58 ms	5.87 ms	5.65 ms
nenorm	4	1.66 ms	1.79 ms	1.68 ms

V tabuľke 5.14 vidíme zmenu v potrebnom čase pre výpoćet modelu pri odstránení normalizácie. Dosiahnuté zrýchlenie bolo 71% pre vstup o dĺžke 2 slov a 70.2% pre vstup o dĺžke 4 slov.

5.4 Kombinácie navrhnutých optimalizácií

Táto sada experimentov sa zaoberá kombináciou navrhnutých optimalizácií. V tejto časti nebudem využívať optimalizáciu pomocou využitia histórie modelu, pretože v predchádzajúcej kapitole s experimentami jednotlivých optimalizácií bol dokázaný jej negatívny vplyv na rýchlosť modelu. V prvom kroku sledujem spoločné využitie aktivačnej funkcie `PReLU`() spolu s predpočítaním matíc pre výpočet skrytej vrstvy, pretože tieto optimalizácie nevyžadujú zmenu vo vstupe a výstupe modelu. Pretože vplyv na jednotlivé časti výpočtu modelu sme sledovali v predchodzej sekcii experimentov budeme sledovať iba urýchlenie celkového výpočtu modelu.

V tabuľke 5.15 vidíme efekt optimalizácie pri použití aktivačnej funkcie `PReLU`() spolu s predpočítanými maticami. Výsledkom tohto experimentu bolo zrýchlenie modelu o 7.3% pre vstupný reťazec s dĺžkov 2 slov a 6.3% zrýchlenie pre vstup s dĺžkou 4 slov.

Ďalší krok sa zaoberá kombináciou prepočítania matíc pre výpočet skrytej vrstvy spolu s nenormalizovaným modelom. Pre tento krok som sa rozhodol z dôvodu, že predpočítanie matíc nemení perplexitu modelu, teda úspešnosť modelu je lepšia ako s modelom využívajúcim aj funkciu `PReLU`().

Tabulka 5.15: Celková rýchlosť modelu s využitím aktivačnej funkcie `PReLU`() spolu s predpočítanými maticami pre výpočet skrytej vrstvy.

Model	Vstup	Min	Max	Avg
baseline	2	5.55 ms	5.81 ms	5.60 ms
projekcia	2	5.18 ms	5.22 ms	5.19 ms
baseline	4	5.58 ms	5.87 ms	5.65 ms
projekcia	4	5.29 ms	5.30 ms	5.29 ms

Tabulka 5.16: Celková rýchlosť nenormalizovaného modelu s využitím predpočítania matíc.

Model	Vstup	Min	Max	Avg
baseline	2	5.55 ms	5.81 ms	5.60 ms
nenorm	2	1.38 ms	1.52 ms	1.41 ms
baseline	4	5.58 ms	5.87 ms	5.65 ms
nenorm	4	1.48 ms	1.59 ms	1.55 ms

Výsledky experimentu s kombináciou nenormalizovaného modelu a predpočítanými maticami je možno vidieť v tabuľke 5.16. Dosiahnuté zrýchlenie bolo 74.8% pre 2-slovný vstup a 72.5% zrýchlenie pre 4-slovný vstup.

Ako posoledný bod experimentov s kombinovanými optimalizáciami je využitý model, ktorý zahŕňa všetky pozitívne optimalizácie. Tento model by mal byť najrýchlejší z modelov.

Tabulka 5.17: Celková rýchlosť nenormalizovaného modelu s využitím aktivačnej funkcie `PReLU`() a predpočítanými maticami pre výpočet skrytej vrstvy.

Model	Vstup	Min	Max	Avg
baseline	2	5.55 ms	5.81 ms	5.60 ms
nenorm	2	1.31 ms	1.43 ms	1.38 ms
baseline	4	5.58 ms	5.87 ms	5.65 ms
nenorm	4	1.36 ms	1.49 ms	1.42 ms

Po dokončení experimentu môžme v tabuľke 5.17 vidieť, že bolo dosiahnuté zrýchlenie 75.3% pre vstup o veľkosti 2 slov a 74.8% zrýchlenie pre vstup o veľkosti 4 slov.

Kapitola 6

Záver

Táto práca sa venovala zrýchleniu výpočtu pravdepodobností pre jazykový model využívajúci doprednú neurónovú sieť. Pomocou optimalizácií sa bolo možné tohto cielu dosiahnuť. Neurónové siete boli zrýchlovalné pomocou zmeny aktivačnej funkcie, implementácie histórie cache do modelu, predpočítaním matíc pre výpočet skrytej vrstvy a odstránením normalizácie modelu. Najúspešnejší model nevyužíva všetky navrhované optimalizácie a to hlavne z toho dôvodu, že pri implementacii jednoduchej histórie modelu nebolo dosiahnuté žiadne zrýchlenie, no naopak nás táto optimalizácia spomaľovala.

Najlepšia optimalizácia dosahovala 75.3% zrýchlenie pre dvojslovný vstup a 74.8% zrýchlenie pre štvorslovný vstup. Tento model bol kombináciou využitia funkcie `PReLU`(), nenormalizovaného modelu a predpočítaných matíc pre výpočet skrytej vrstvy.

Výsledky tejto práce je možné považovať za úspešné, no stále je množstvo miest na zlepšenie ako implementácia v jazyku C. Využitie jazyka C by napomohlo optimalizácii s využitím histórie, kde by bolo vyhľadávanie v histórii urýchlené z dôvodu nahradenia pomalého cyklu. Taktiež by bolo možné urýchliť indexácie pri optimalizáciach s históriou a predpočítanými maticami.

Časť tejto práce bola prezentovaná na konferecii Excel@FIT 2018 [7].

Naďalej by som chcel v práci zdokonaliť programovú časť, ktorá pri experimentoch vyžadovala zásahy do kódu programu. Taktiež by som chcel vyhľadať možnosti zlepšenia perplexity modelu, nakoľko dve optimalizácie urýchlili model za cenu jeho presnosti.

Literatura

[1] Bengio, Y.; Ducharme, R.; Vincent, P.; aj.: A neural probabilistic language model. *Journal of machine learning research*, ročník 3, č. Feb, 2003: s. 1137–1155.

[2] Brown, P. F.; Desouza, P. V.; Mercer, R. L.; aj.: Class-based n-gram models of natural language. *Computational linguistics*, ročník 18, č. 4, 1992: s. 467–479.

[3] Chen, S. F.; Goodman, J.: An empirical study of smoothing techniques for language modeling. In *Proceedings of the 34th annual meeting on Association for Computational Linguistics*, Association for Computational Linguistics, 1996, s. 310–318.

[4] Devlin, J.; Quirk, C.; Menezes, A.: Pre-computable multi-layer neural network language models. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, 2015, s. 256–260.

[5] Gutmann, M. U.; Hyvärinen, A.: Noise-contrastive estimation of unnormalized statistical models, with applications to natural image statistics. *Journal of Machine Learning Research*, ročník 13, č. Feb, 2012: s. 307–361.

[6] Huang, Y.; Sethy, A.; Ramabhadran, B.: Fast Neural Network Language Model Lookups at N-Gram Speeds. *Proc. Interspeech 2017*, 2017: s. 274–278.

[7] Labaš, D.: Akcelerácia neurónovej siete pre jazykové modelovanie. *Excel@FIT, studentská konference inovací, technologií a vědy v IT*, 2018.

[8] Marcus, M. P.; Marcinkiewicz, M. A.; Santorini, B.: Building a large annotated corpus of English: The Penn Treebank. *Computational linguistics*, ročník 19, č. 2, 1993: s. 313–330.

[9] Paszke, A.; Gross, S.; Chintala, S.; aj.: Automatic differentiation in PyTorch. In *NIPS-W*, 2017.

[10] Xu, B.; Wang, N.; Chen, T.; aj.: Empirical evaluation of rectified activations in convolutional network. *arXiv preprint arXiv:1505.00853*, 2015.